Calcul de la taille optimale des paquets pour les algorithmes macro-pipelines

Pierre Ramet *
LaBRI (URA CNRS 1304), Université Bordeaux I, 33 405 Talence Cedex, France
E-mail: ramet@labri.u-bordeaux.fr

1. Introduction et Problématique

L'utilisation des machines parallèles à mémoire distribuée apporte un gain important en performances et en taille mémoire mais amène en contre-partie un surcoût en communications. Afin d'obtenir des programmes performants et extensibles, il convient de cacher ce surcoût. Plusieurs solutions existent. La première consiste en un choix judicieux de la distribution des données qui réduira au maximum le nombre et la taille des communications. De plus, si les dépendances le permettent, on essaiera de lancer les communications au plus tôt et de manière asynchrone. Pendant l'exécution de ces communications, on effectuera d'autres calculs, sans rapport avec l'échange de données. Cette optimisation n'est malheureusement pas toujours réalisable à cause des dépendances dans les calculs. Suivant ces dépendances, on peut également parfois avoir des schémas de type pipelines comme celui donné dans la figure 1 (A). Il y a une sequentialité dans l'exécution. Le processeur P1 doit attendre que le processeur P0 ait fini de calculer pour recevoir les résultats du calcul et commencer à travailler. Une fois qu'il a terminé, il envoie les données résultats au processeur P2 qui peut, à son tour, travailler sur les données reçues. Le temps total est plus important que le temps séquentiel puisque l'on introduit le surcoût des communications. Une première solution consiste à commencer les communications au plus tôt, c'est à dire dès qu'un processeur a calculé une donnée. Pour chaque donnée calculée, un envoi est effectué et les processeurs commencent leur travaux dès que possible. On a donc un pipeline à grain fin (B sur la figure 1). Cette solution introduit malgré tout un surcoût dû aux temps d'initialisation des communications. Ces temps sont toujours plus importants que le coût d'envoi d'un élément et il en résulte que le temps total peut être encore plus important que le temps séquentiel. Un compromis consiste donc à trouver le grain de calcul (et de communication) qui minimisera le temps total d'exécution. On aura alors un pipeline à grain moyen (C sur la figure 1).

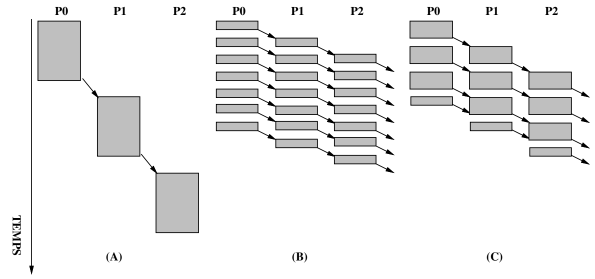

FIG. 1 - . Recouvrement des communications grâce à un pipeline.

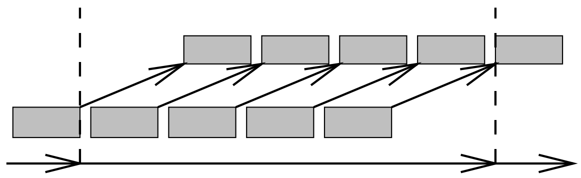

FIG. 2 - . Recouvrement calcul/communication.

Dans le cas linéaire, le calcul du grain est facile à calculer [2]. Ce n'est plus le cas dès que la complexité des calculs et des communications n'est plus une fonction linéaire de la taille des paquets. Le but de cet article est donc de présenter nos résultats sur le calcul de la taille optimale de paquets dans le cas général. On trouvera une version tendue dans [3].

Des travaux existent déjà sur le sujet, en particulier dans le domaine de la compilation automatique. Le cas linéaire qui a motivé cette étude est présenté dans [2]. Les résultats sont utilisés dans le cadre de l'utilisation de la bibliothèque LOCCS [1]. Dans [4], la taille optimale des paquets est

*Ce travail est soutenu par le GDR-PRS

calculée pour des nids de boucles DOACROSS généraux. Les complexités issues de tels calculs sont linéaires dans la taille des données. Les optimisations pour le compilateur Fortran D sont présentées dans [6]. Ici encore, les cas simples liés à la compilation de nids de boucles sont donnés. L'auteur insiste sur le fait qu'une analyse plus précise serait nécessaire pour obtenir des performances optimales. Enfin, dans [5], le cas du nid à deux boucles est présenté pour des réseaux de stations, mais cette fois avec un calcul reporté à l'exécution, qui permet un ajustement plus précis, puisqu'il dépend de la charge de la machine lors de l'appel.

2. Description du modèle retenu

Dans cet article, nous présentons un modèle théorique pour le recouvrement calcul/communication ainsi qu'une méthode pour calculer la taille optimale des paquets qui maximise le recouvrement.

On considère un schéma macro-pipeline à 2 processeurs. Le premier effectue un calcul avant de communiquer ses résultats, utilisés par le second processeur pour son traitement. Soit L la taille du macro-pipeline. On le découpe en μ paquets de taille ν (voir figure 2).

On appelle "calcul avant" le travail qui doit être effectué sur un paquet de taille ν avant de communiquer les résultats correspondants. Son temps d'exécution est donné par

$$Tcalc_A(\nu, L) = R_a(L)\, f_a(\nu)\, \tau_a.$$

$f_a(\nu)$ représente la complexité de ce travail qui dépend de son implémentation et de la distribution des données, $R_a(L)$ représente la constante de la complexité du calcul qui peut dépendre de la taille du macro-pipeline, τ_a représente le temps élémentaire du "calcul avant" (sur un élément du paquet).

De même, le "calcul après" représente le travail qui doit être effectué à l'aide des résultats communiqués.

$$Tcalc_B(\nu, L) = R_b(L)\, f_b(\nu)\, \tau_b.$$

Enfin, soit $Tcomm(\nu)$ le temps de communication d'un paquet de taille ν, son expression peut dépendre du type de communication utilisé (diffusion, échange total, ...) et de l'architecture du calculateur.

Le temps total d'exécution du macro-pipeline est donné par

$$Ttotal(\nu, L) = Tcalc_A(\nu, L) + \tfrac{L}{\nu}\ \max\{Tcalc_A(\nu, L), Tcomm(\nu), Tcalc_B(\nu, L)\} + Tcalc_B(\nu, L).$$

Sous les contraintes $\left\{\begin{array}{l} Tcalc_A(\nu, L) < Tcomm(\nu) \\ Tcalc_B(\nu, L) < Tcomm(\nu) \\ 0 < \nu < L \end{array}\right.$

la minimisaton de $Ttotal(\nu, L)$ revient à minimiser $Tcalc_A(\nu, L) + \frac{L}{\nu}\, Tcomm(\nu) + Tcalc_B(\nu, L)$.

Pour calculer la taille optimale on résout d'abord, soit de manière directe, soit à l'aide de méthodes approchées, ν_{min} réalisant $\frac{\partial}{\partial\nu} Ttotal(\nu, L) = 0$. Pour la plupart des complexités surlinéaires de calculs "avant" et "après" et des types de communication, on peut vérifier l'existence et l'unicité de la solution en montrant que $\frac{\partial^2}{\partial\nu^2} Ttotal(\nu, L) \geq 0$.

On calcule ensuite la taille de paquets, ν_{bal}, qui équilibre la contrainte $Tcalc(\nu, L) = Tcomm(\nu)$ avec $Tcalc = max(Tcalc_A, Tcalc_B)$. Pour $\nu > \nu_{bal}$, $Ttotal(\nu, L)$ est une fonction croissante et admet une valeur minimale en ν_{bal}.

La taille optimale est alors donnée par : $\nu_{opt} = \min(L, \max(1, \nu_{min}), \max(1, \nu_{bal}))$.

On retrouve, pour un calcul en $\theta(\nu)/\theta(\nu)$, le résultat connu [2].

Prenons maintenant l'exemple d'un calcul en $\theta(\nu^2)/\theta(\nu)$: $\left\{\begin{array}{l} Tcalc_A(\nu) = R_a\, \nu^2\, \tau_a \\ Tcalc_B(\nu) = R_b\, \nu\, \tau_b \\ Tcomm(\nu) = \beta + \nu\, \tau \end{array}\right.$

on a $\nu_{bal} = \frac{\tau + \sqrt{\tau^2 + 4\, R_a\, \tau_a\, \beta}}{2\, R_a\, \tau_a}$ et on montre que l'on trouve 3 solutions (réelles ou complexes) mais dont une seule est réelle positive

$$\left\{ \begin{array}{l} \nu_1 = \sqrt[3]{\frac{-q+\delta}{2}} + \sqrt[3]{\frac{-q-\delta}{2}} - \frac{A}{3} \\ \nu_2 = \sqrt[3]{\frac{-q+\delta}{2}} \cdot j + \sqrt[3]{\frac{-q-\delta}{2}} \cdot j^2 - \frac{A}{3} \\ \nu_3 = \sqrt[3]{\frac{-q+\delta}{2}} \cdot j^2 + \sqrt[3]{\frac{-q-\delta}{2}} \cdot j - \frac{A}{3} \end{array} \right. \quad \text{avec} \left\{ \begin{array}{l} p = -A^2/3 \\ q = 2\, A^3/27 - B \\ \delta^2 = \frac{4\, p^3 + 27\, q^2}{27} \end{array} \right. \quad \text{et} \left\{ \begin{array}{l} A = \frac{R_b\, \tau_b}{2\, R_a\, \tau_a} \\ B = \frac{L\, \beta}{2\, R_a\, \tau_a} \end{array} \right.$$

On voit que le calcul de la taille optimale des paquets peut-être coûteux, en particulier si on utilise des méthodes itératives (de type Newton) pour obtenir une solution approchée dans le cas où le calcul direct est impossible. Il est donc intéressant d'envisager différentes techniques pour éviter de recalculer la taille optimale des paquets lorsque l'on fait varier la taille d'un même schéma macro-pipeline.

On peut calculer le domaine de validité pour lequel la taille optimale reste inchangée lorsque l'on fait varier la taille du macro-pipeline. Pour cela on calcule l'écart l maximal tel que $|\nu_{min}(L+l) - \nu_{min}(L)| < 1$. On combine cette technique avec l'utilisation d'une table de hachage dans laquelle on enregistre la taille optimale et son domaine de stabilité pour les différentes tailles du schéma macro-pipeline déjà calculées.

3. Expérimentations

Nous avons choisi une implémentation basée sur les LOCCS [2] fournissant des primitives de gestion du recouvrement. La version actuelle de la bibliothèque OPIUM (Optimal Paquet sIze compUtation Methods) permet le calcul de la taille optimale des paquets pour des algorithmes macro-pipelines de complexité linéaire et quadratique, elle contient aussi une primitive générale, utilisant un solveur de type Newton, et acceptant une complexité quelconque.

L'exemple choisi est un algorithme de factorisation de type LU par colonnes. L'implémentation des calculs "avant" et "après" est écrite de sorte qu'ils aient la même masse de calcul. Ainsi, on peut espérer obtenir un gain de recouvrement optimal qui diminue de moitié le temps de factorisation. Les mesures de performances sont réalisées sur un calculateur Intel Paragon avec les paramètres: $Tcomm(\nu) = 3\;10^{-5} + 2\;10^{-9}\;\nu$; $\quad \tau_A = \tau_B = 5.2\;10^{-7}$.

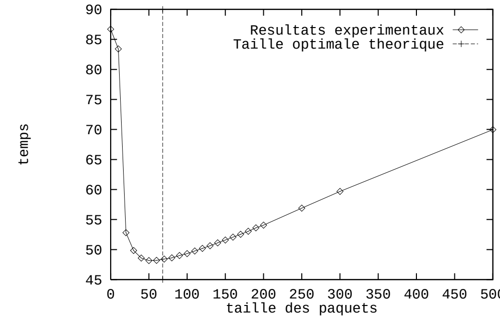

FIG. 3 - . Temps de factorisation d'une matrice de taille 1024 pour différentes taille de paquets.

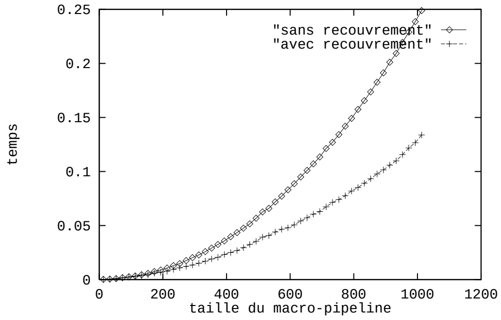

FIG. 4 - . Temps passé dans chaque macro-pipeline de la factorisation d'une matrice de taille 1024 (avec et sans recouvrement).

La taille prédite et calculée à partir de notre modèle s'avère être très proche de la valeur obtenue expérimentalement (voir figure 3); et de plus elle est, comme prévue, *constante* pour chaque macro-pipeline jusqu'à l'intervention de la contrainte; elle n'est donc calculée qu'une fois. Les figures suivante décrivent la qualité du recouvrement. On voit sur la figure 4 que le recouvrement est d'autant plus efficace que la taille du macro-pipeline est importante. Le gain apporté par le recouvrement décroit avec le nombre de processeurs, cela est dû au fait que le travail exécuté par chaque processeur devient trop faible pour que le recouvrement joue un rôle significatif (voir figure 5). Enfin la courbe du gain, figure 6, montre que pour une matrice de taille suffisament grande, on atteint

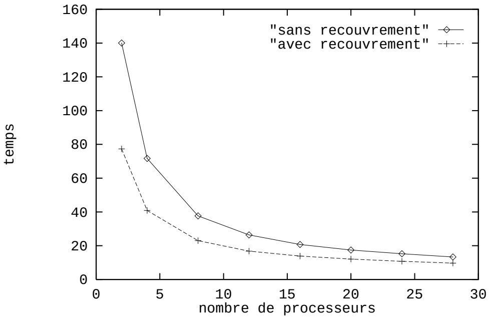

FIG. 5 - . Temps de factorisation (avec et sans recouvrement) d'une matrice de taille 960 suivant le nombre de processeurs alloués.

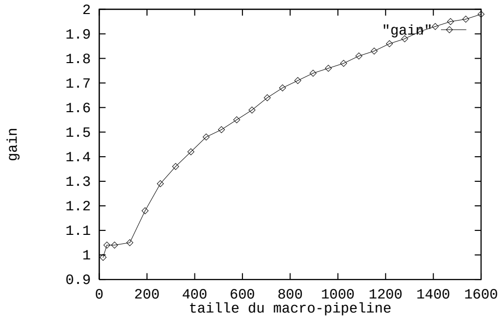

FIG. 6 - . Rapport entre le temps de factorisation avec et sans recouvrement suivant la taille de la matrice.

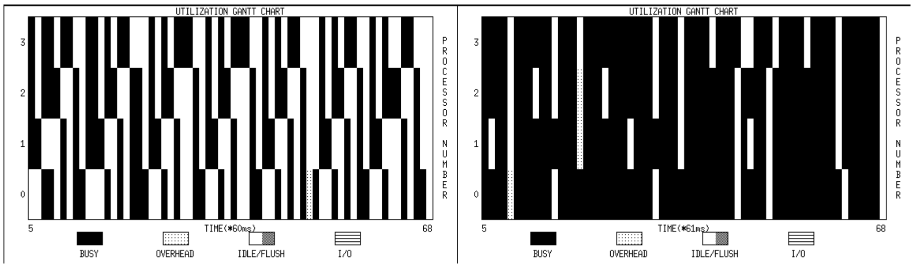

FIG. 7 - . Diagramme de Gantt. (en blanc l'inactivité et en noir l'activité des processeurs)

pratiquement le recouvrement maximal (gain de 50%). Quelle que soit la taille de la matrice, on voit également que le surcoût engendré par la gestion du recouvrement est négligeable (gain > 1). Au niveau du palier, pour $L < 100$ on ne découpe pas ($\nu_{opt} > L$).

La figure 7 représente l'activité des 4 processeurs alloués pour la factorisation d'une matrice de taille 512 sans et avec recouvrement. Ce graphique montre la bien meilleure utilisation du parallélisme avec les mécanismes de recouvrement.

4. Conclusion

Ces résultats montrent l'efficacité d'une bonne gestion du recouvrement, non seulement sur des schémas macro-pipelines, mais également sur une application complète. On retrouve, de manière expérimentale, les résulats théoriques du modèle. D'autres expérimentation sont en cours pour valider le modèle sur d'autres algorithmes. En particulier, il sera intéressant d'intégrer ces techniques sur des problèmes de nature irrégulière et d'adapter le modèle pour prendre en compte l'utilisation des BLAS, l'objectif étant d'offrir ces mécanismes dans un processus de compilation automatique.

Bibliographie

1. F. Desprez. A Library for Coarse Grain Macro-Pipelining in Distributed Memory Architectures. In *IFIP 10.3 Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 365–371. Birkhaeuser Verlag AG, Basel, Switzerland, 1994.
2. F. Desprez. *Procédures de Base pour le Calcul Scientifique sur Machines Parallèles à Mémoire Distribuée.* PhD thesis, Institut National Polytechnique de Grenoble, January 1994. LIP ENS-Lyon.
3. F. Desprez, P. Ramet, and J. Roman. Optimal grain size computation for pipelined algorithms. submitted to europar'96. 1996.
4. H. Ohta, Y. Saito, M. Kainaga, and H. Ono. Optimal Tile Size Adjustement in Compiling General DOACROSS Loop Nests. In ACM Press, editor, *International Conference on Supercomputing*, pages 270–279, Barcelona, Spain, July 1995. ACM SIGARCH.
5. B.S. Siegel and P.A. Steenkiste. Controlling Application Grain Size on a Network of Workstations. In *Supercomputing'95*, 1995.
6. C.W. Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines.* PhD thesis, Rice University, January 1993.